IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re: Kerimovska et al.   Confirmation No.: 9239
Serial No.: 10/539,238   Art Unit: 2626
Filed: April 10, 2006   Examiner: Jakieda R. Jackson

For: **DEVICE FOR GENERATING SPEECH, APPARATUS CONNECTABLE TO OR INCORPORATING SUCH A DEVICE, AND COMPUTER PROGRAM PRODUCT THEREFOR**

DECLARATION PURSUANT TO 37 C.F.R. §1.131

Sir:

We, Nercivan Kerimovska, Gunnar Klinghult, and Anna Tomasson, hereby declare and say that:

1. We are the named inventors of the subject matter of the inventions disclosed and claimed in Application Serial No. 10/539,238, filed on April 10, 2006, which is a 35 U.S.C. §371 national phase application of PCT International Application No. PCT/EP2003/012879, having an international filing date of November 14, 2003 and claiming priority to European Patent Application No. 02445177.5, filed December 16, 2002.

2. Prior to December 11, 2002, we conceived and reduced to practice, in Sweden (a WTO member country), the subject matter of at least independent Claims 1, 20, and 39, which are currently pending in the patent application.

3. In support of the above statement of Section 2, we hereby submit as **Appendix A** a copy of a Draft Specification by inventors Anna Tomasson and Nercivan Kerimovska that was finalized on November 25, 2002.

4. The Draft Specification of **Appendix A** establishes that the subject matter of at least independent Claims 1, 20, and 39 was reduced to practice on or before December 11, 2002. In particular, Figure 4.1 of the Draft Specification illustrates a prototype that provides the functionality recited by at least independent Claims 1, 20, and 39.

5.   In summary, our statements herein and the documents we have concurrently submitted establish actual reduction to practice of the invention in a WTO member country prior to December 11, 2002.

6.   I hereby declare that all statements made herein of my own knowledge are true, and that all statements made on information and belief are believed to be true. I further declare that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.
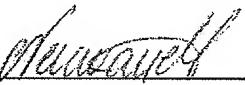
_____
Nercivan Kerimovska

_090331_
Date
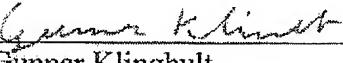
_____
Gunnar Klinghult

_24/3 - 09_
Date

_____
Anna Tomasson

_090323_
Date

Created by: Anna Tomasson
Nercivan Kerimovska
Date: 2002-11-25
Version: 0.9

Contents

1 Introduction

Mobile phones have been a part of our lives for over two decades and have experienced an explosive technical development. The main function is still, of course, to set up phone calls but today there are millions of other possibilities. The technical progress has made it possible to make the phones cheaper and smaller but more usable. Mobile phones have become user-friendlier with polyphonic sounds and larger colour displays. They have also been made for use in different kinds of environments.

The phones have been developed with focus to fit different kinds of people, for example the possibility for businessmen to make conference calls and teenagers to play games. But still there is one category that is excluded, the ones with visual impairment. A larger display with different colours is not enough, but with use of other senses these people may use a phone like anyone else. The only requirement is that the phone can talk. Is that possible?

An ordinary service like SMS has been completely ruled out since it has been impossible to read them. A mobile phone adapted for this group must among other things help the user by reading SMS and maybe most important of all by reading the menus to help locate while browsing.

1.1 The Goal and How to Achieve it

The goal with this master's thesis was to develop a prototype for text-to-speech (TTS) conversion, converting menus and SMS. The prototype is required to be able to be attached to the mobile phones T68i system connector as an accessory to the phone. This requires that the prototype is of proportional size i.e. not much larger than the mobile phone itself and that it is not ungainly. The goal was clear, but how was it approached?

The idea was to use an already existing TTS circuit and combine it with a microcontroller. The block diagram in Figure 2.1 gives an overview of the prototype.
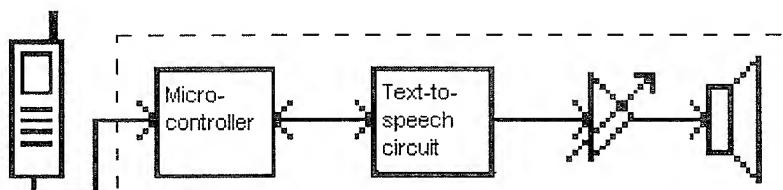
Figure 2.1 Block diagram over the main blocks i.e. phone, microcontroller, TTS circuit, amplifier and loudspeaker.

The parts within the dashed square is intended to represent the accessory. The idea was to let the microcontroller get the text to be converted from the mobile phone and pass it through to the TTS circuit. The TTS circuit converts the text to audio signals and sends them via an amplifier to a loudspeaker. The phone was supposed to communicate with the accessory with attention (AT) commands. AT commands and certain protocols are used for communication between the phone and accessories connected to the system contact of the phone.

1.2 Report Outline

The outline of this report is as follows:

Chapter 1: In this chapter an introduction to the report is given.

Chapter 2: This chapter describes the underlying research and contains the theory about the circuits to be used. Furthermore it contains study of the software design and the interfaces needed.

Chapter 3: This chapter describes the different parts of the implementation process.

Chapter 4: In this chapter the tools used during the development of the accessory are presented.

Chapter 5: The result of the thesis is discussed in this chapter and also the limitations of the accessory are described.

Chapter 6: This chapter discusses other possibilities with this application and different ways to implement it.

2 Research

To achieve the goal a proper research must be done, an investigation of what hardware to be used and how the software should be designed. Also the interfaces between the circuits and how they should be connected must be considered before a final decision of the shape of the prototype can be made. An important step in this procedure is a careful study of the data sheets and the application notes.

The development process was planned to consist of five steps:
1. **Research** – pre-study of how the development will proceed, decision of which circuits to be used and study of documentation/data sheets.
2. **Wired prototype** – a primer coupling is wired for expected hardware behavior and for basic error detection.
3. **The microcontroller software** – initialization of the circuits and implementation of the interfaces and the functionality required,
4. **The phone software** – implementation of needed software in the already existing phone software.
5. **Circuit board** – computer aided design (CAD) of the printed circuit board (PCB), etching and soldering.

2.1 The Hardware

The Atmel AVR 90S8515 was the pre-defined microcontroller to be used in this project and on the basis of this the other circuits were chosen.

The Winbond WTS701 was selected among a number of different, existing TTS circuits on the market, since it had the best speech quality and seemed to be relatively easy to mount and use. This circuit has a differential output designed to drive an 8Ω speaker and apart from this also a tele socket for a headphone or an external loudspeaker was suggested. The tele socket works in such way that when an external loudspeaker is connected to it, the built-in loudspeaker is disconnected. To get the right volume level an amplifier is also needed. Figure 2.1 shows the interaction between the different hardware blocks.
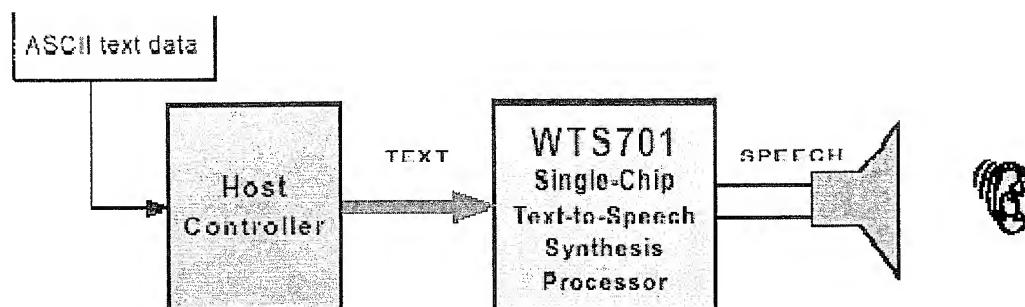

Figure 2.1 An overview of the HARDWARE coupling.

For customization it should be possible to change the volume and therefore buttons are needed.

The accessory must also have a system connector to be able to be connected to the MS.

If possible it is convenient to let the MS supply the accessory with power. According to the datasheets of the parts planned to be used, the maximum voltage level is limited by the data voltage level between the MS and the accessory to +2.7V. Since this level is below the amount of power that the MS may deliver to an external device, +3.8V, it is possible to supply the accessory from this source. To convert +3.8V to +2.7V a voltage converter is also necessary.

2.1.1 The Microcontroller

The AT90S8515 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. The most important features of the microcontroller is the 8K bytes of In-System Programmable Flash, 32 general-purpose I/O lines, 32 general-purpose working registers, internal and external interrupts, a programmable serial Universal Asynchronous Receiver and Transmitter, UART, and a port for the Serial Peripheral Interface, SPI. The On-chip In-system Programmable Flash allows the program memory to be reprogrammed In-System through SPI. This means that it is possible to flash the microcontroller with new software without removing it from the circuit board.

The microcontroller has four different ports named A, B, C and D that consists of 8 pins each and serves as 8-bit bi-directional I/O ports. Some of the pins may have other functionalities that are enabled by writing to the corresponding register. The registers control the behavior of the microcontroller and by writing to these the functionality of the microcontroller is customized. Port A and C can be used for communication with an external memory. Port B may serve as the SPI interface and port D for UART communication.

The microcontroller has a corresponding emulator, ATICE200, which is pin compatible and can be used for developing a wired prototype. The emulator simulates and behaves like the microcontroller but is controlled by a computer. It makes it possible to follow each command and it shows how the different ports, pin values and register changes during execution. It is also possible to single-step i.e. execute one line at a time to see if the behaviour is the expected, see chapter XXX.

This is a great advantage that aims to easier debugging, and therefor planned to be used in this project. Since the wired prototype, which contains hole mounted circuits, will not have an acceptable size also a prototype with surface mounted circuits must be developed.

2.1.2 The Text-to-Speech Circuit

The TTS circuit, Winbond WTS701, is a high quality, fully integrated and single-chip solution that is ideal for this application, mainly because it supports SMS and has a general modifiable abbreviation list.

2.1.2.1 The Text-to-Speech Mechanism

The TTS component of the system consists of three principal blocks:

- Text normalization
- Word to phoneme conversion
- Phoneme mapping
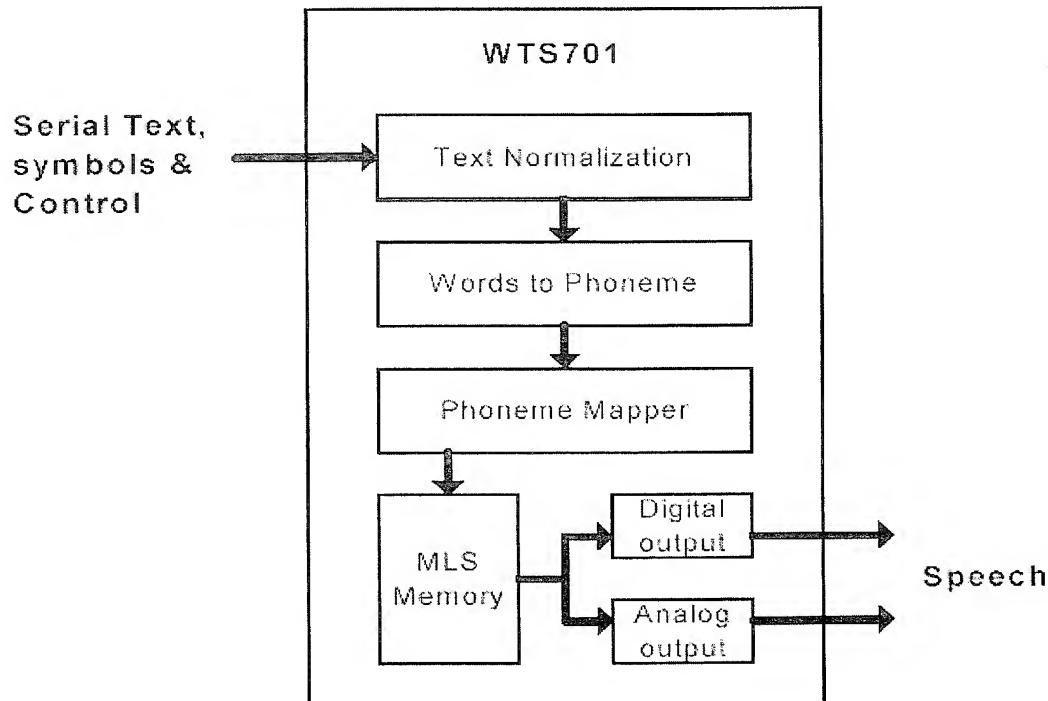
Figure 2.2 TTS system process flow.

2.1.2.2 Text Normalization

This is the process of translating the incoming text to pronounceable words. It expands abbreviations and translates numeric strings to spoken words.

The abbreviation list can be modified. This enables flexibility of adding abbreviation specifically for the text, either by the developer or by the end user to customize the product. Even the unique characters of SMS are supported, meaning that icons (such as smilies) will be replaced by its true meaning. This means that a SMS containing abbreviations and icons will be recited correct.

2.1.2.3 Words-to-Phoneme Mapping

When the text has been translated into pronounceable words, the system must decide how to pronounce them. This is not an easy task since most languages is made of a complicated set of rules for pronouncing. To handle this task the system works with a combination of rule based processing and exception processing. An example of a word that is pronounced differently depending on the rest of the sentence is "read". It can either be pronounced 'red' or 'reed'.

2.1.2.4 Phoneme Mapping

No dictionary is unlimited, which means that the system must divide words into sub-words that it recognizes from the stored list of words in the internal memory. The dividing or splitting must be done at appropriate phonetic boundaries to achieve high quality concatenation. Once a sub-word unit is determined, the inventory is searched to determine if a match is present. A matching weight is assigned to each match depending on how closely the phonetic context matches. Each sub-word has a left and right side context to match as well as the phoneme string itself. If no suitable match is found in the inventory, then the sub-word is further split in a tree-like manner until a match is found. The splitting tree is processed from left to right and each time a successful match occurs, the address and duration of the match in the corpus is placed in a queue of phonetic parts to be played out in the audio interface.

2.1.3 The Amplifier

The LM4894MM is a fully differential audio power amplifier capable of delivering 1W of continuous average power to an 8Ω load. The gain is set with two resistors, R_F and R_I, for each input to:

$$Gain = -\frac{R_F}{R_I}$$

It is important to match the input resistors, R_I, and the feedback resistors, R_F, to each other for best amplifier performance.

To save power it is possible to set the amplifier in shutdown mode. This is done by toggling its Shutdown Select pin to the same state as its Shutdown Mode pin.

2.1.4 The Voltage Regulator

The MAX8863 is a low-dropout linear regulator that operates from +2.5V to +6.5V input voltage. The output voltage is either preset or adjusted by two external resistors as a voltage divider by the following equation:

$$V_{out} = V_{set}\left(1 + \frac{R_1}{R_2}\right)$$

V_{set}=1.25V and to optimize power consumption R_2 is set to 100kΩ.

2.1.5 The Buttons

Every press on one of the buttons should make the microcontroller either increase or decrease the volume. The microcontroller can easily detect a press by having the buttons connected to its interrupt pins (INT0 and INT1) and letting them generate interrupts.

2.1.6 The System Connector

To be able to connect the accessory to the MS a system connector, BOX Plug RPV 899 06, is needed. The system connector interface consists of audio signals, two serial channels, power leads and the analogue and digital ground leads. For an overview of the connector and its pins, see Figure 2.3 and Table 2.1.
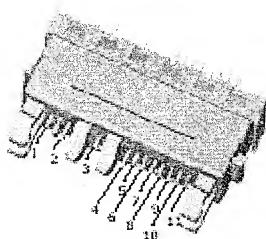
Figure 2.3 Full System Connector, accessory side.

Pin	Signal	Function	I/O 52
1	DCIO	DC + pole for charging of the phone battery and external accessory powering	I/O
2	DGND	Digital GrouND and DC Return	-
3	VPPFLASH	Flash Memory Vpp / Service	I
4	AGND	Audio Signal GrouND. 0 V reference.	-
5	CFMS/PHFS	Accessory Control From Mobile Station serial bus (ACB) communication / Portable HandsFree Sense.	O/I
6	CTMS	Accessory Control To Mobile Station serial bus (ACB) communication	I
7	DFMS	Data From Mobile Station, serial bus communication	O
8	DTMS	Data To Mobile Station, serial bus communication	I
9	CTS/ONREQ	Clear To Send / Mobile Station On REQuest	O/I
10	AFMS / RTS	Audio From Mobile Station / Ready To Send	O/I
11	ATMS	Audio To Mobile Station	I

Table 2.1 The pins of the connector and their corresponding signals and functions.

2.2 The Software

The microcontroller is the circuit that will control the entire accessory and the communication with the phone and must therefore be programmed to achieve the wanted behavior. Since the microcontroller can not control the phone, programming in the phone is also required. Both the code in the microcontroller and in the phone is written in C.

The initial plan was to use AT commands to let the accessory communicate with the phone by sending requests for the text strings. This proved to be impossible since no AT commands for sending text from the phone exists. To circumvent the problem it is possible to let the phone print the data to the system bus in the same way as it does when it is being logged. To log the phone a NOR plug is connected between the phone system connector and a computer. The NOR plug sets the phone in so called service mode, which is necessary for the phone to be able to put data on the system bus. To place data on the bus an ordinary *printf()* might be used.

2.2.1 The Microcontroller Software

To be able to control the accessory the microcontroller must be initiated and customized for this purpose. This is done by setting its registers, defining which pins to be inputs/outputs and enabling the alternative functions.

The microcontroller software must also initiate the TTS circuit and include a set of communication commands and functions for the SPI and UART interfaces. It must also handle and react on possible interrupts.

2.2.2 The Phone Software

The phone must support the accessory in the way that it must send the data to it. To put text strings on the system bus a few lines of code must be added to affected modules in the general T68i code. The main purpose is to extract the text string from the text id and send it to the accessory with a *printf()*. The text id is used to point out the text string that is to be read.

2.3 The Interfaces

The different blocks in the system need the right interfaces to communicate properly with each other. The interface between the phone and the microcontroller consists of a UART while the microcontroller and the TTS circuit communicate via SPI.

The data (ASCII characters), sent from the phone to the accessory, is intended to be received and handled by the microcontrollers Universal Asynchronous Receiver and Transmitter, UART, and then sent to the TTS circuit. The communication between the TTS circuit and the microcontroller is supposed to be handled by the Serial Peripheral Interface, SPI. The data flow is illustrated in Figure 2.4.
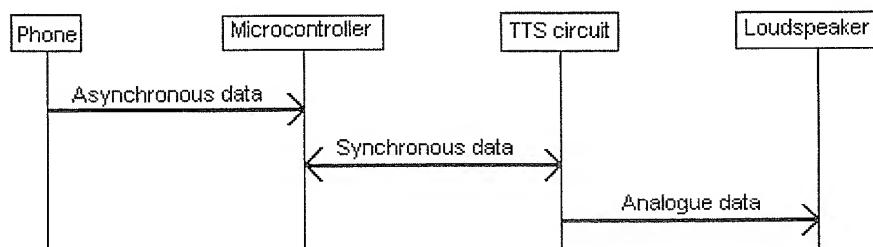

Figure 2.4 Data flow diagram.

2.3.1 The UART

The microcontroller features a full duplex UART with the main features:

• Baud rate generator that can generate a large number of baud rates

- High baud rates at low XTAL frequencies
- 8 or 9 bits data
- Noise filtering
- Overrun detection
- Framing error detection
- False start bit detection
- Interrupts

2.3.1.1 Data Transmitter

Data transmission is initiated by writing the data to be transmitted to the UART I/O Data Register (UDR). The data is then transferred from UDR to the transmitter shift register and shifted out on the UART output line (TX pin).

Since it was not possible to use AT commands there is no need to transfer data to the phone and the data transmitter is therefore of no interest in this application.

2.3.1.2 Data Receiver

The receiver samples the incoming signal on the UART input line (RX pin) at a frequency 16 times the baud rate. The baud rate can be set in the UART BAUD Rate Register (UBRR), see Appendix XXX. Every bit received will be sampled 16 times and sample 8, 9 and 10 will decide the logical value of the bit. An incoming sample of logical "0" will be interpreted as the falling edge of a start bit. After a valid start bit sampling of the data bits is performed and finally a stop bit is received. Whether or not a valid stop bit is detected, the data is transferred to UDR, a flag is set and the UART Receive Complete interrupt is executed. UDR is in fact two physically separate registers, one for transmitted data and one for received data.

2.3.2 The SPI

The SPI is a serial interface developed by Motorola and is used primarily for synchronous serial communication of host processor and peripheral circuits. The standard SPI system is made of one master and one or more slaves. A device may be in master or in slave mode, but not in both at the same time. The master device provides the clock signal and determines the state of the chip select lines i.e. it activates the slave it wants to communicate with. This means there is one master, while the number of slaves is only limited by the number of chip selects. Since an ordinary I/O pin may be used as chip select, it is the master that limits the number of slaves. In this application the microcontroller is the master and the TTS circuit the only slave.

The basic principle behind the SPI is an ordinary shift register. Command codes as well as data are serially transferred, clocked into the shift register and then internally available for parallel processing. The length of the shift register is not fixed and may differ from device to device, but normally it is space for 8 bits.

The SPI requires two control lines, Chip Select (CS) and Serial Clock (SCLK), and two data lines, Serial Data In (SDI) and Serial Data Out (SDO). Motorola names these lines Master-Out-Slave-In (MOSI) and Master-In-Slave-Out (MISO). The chip select line is named Slave-Select (SS). The majorities of the SPI devices provide these four lines, see Figure 2.5.
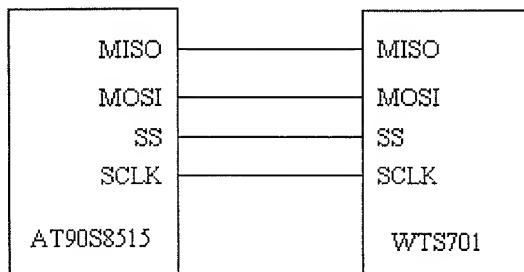
Figure 2.5 The different SPI lines.

Because there is no official specification what SPI is and what not, it is necessary to consult the data sheets to see the specific demands of clock synchronization i.e. the permitted clock frequencies and the type of valid transitions. Although not specified, in practice four modes are used for transitions. These four modes are the combination of Clock Polarity (CPOL) and Clock Phase (CPHA). It is important that these agree with both master and slave.

2.3.2.1 The Communication Protocol

The microcontroller can control the TTS circuit by sending commands via the SPI interface. Any transaction always starts by sending a command. The command consists of a command word and a command byte. At the same time the command is shifted out on the MOSI line, the TTS circuit shifts out its status register on the MISO line.

There are four different types of commands, depending on the interaction with the WTS701. Class 0 commands are commands that will execute irrespective of the state of the circuit. Class 1 commands require interpretation by the internal firmware. Class 2 commands have associated data and finally class 3 commands have data to return to the host.

2.3.2.2 The Communication Signals

A SPI transaction is initiated when the master writes data to its SPI data register, the SPDR register. This register contains 8 bits. Writing to this register starts the SPI clock generator, SCLK. The only time this clock is active is during a SPI transaction. The slave responds to a command when the CS is low and addressed by an active low signal on its SS pin. Since this system only contains one slave, the TTS circuit, the SS pin of the master, i.e. the microcontroller, is not used. Under this condition, the TTS circuit accepts data on the MOSI input, which is clocked in on rising edges of the SCLK signal. The data written in SPDR is shifted out on the master MOSI line and in to the MOSI line of the slave. After shifting one byte the SPI generator stops, sets the end-of-transmission flag (SPIF) and an end-of-transmission interrupt is executed. This interrupt is enabled by setting the SPIE bit in the SPCR register. An SPI transaction is finished when SS is returned to the high condition. The two shift registers in the master and the slave can be considered as one distributed 16-bit circular shift register.

BILD SID 48 I DATABLADEN FÖR AT90S8515

When data is shifted from the master to the slave, data is also shifted in the opposite direction simultaneously. This means that during one shift cycle, data in the master and the slave are interchanged. The system is single-buffered in the transmit direction and double-buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI data register before the entire shift cycle is completed. When receiving data, a received byte must be read from the SPI data register before the next byte has been completely shifted in. Otherwise the first byte is lost.

To enable the SPI on the microcontroller the SPE bit in the SPI control register SPCR must be set and to select master mode the MSTR bit in the same register is set. See Appendix XX for further information of the register settings.

3 Implementation

With the starting point of the parts decided to be used a wired prototype was created and on the basis of this the final circuit board. The reason for doing this is that it will simplify error detection. This makes it easier to measure the signals with oscilloscope and to detect errors in the coupling. It is also easier to modify the original train of thought.

The circuits on the wired prototype are all hole mounted circuits, except for the TTS circuit that has to have a special socket. The circuits on the final circuit board are all surface mounted.

3.1 The Hardware

The hardware used in the wired prototype is the same as used in the final circuit board except for the microcontroller. The microcontroller used on the wired prototype is of different shape and pin amount than the one used on the circuit board.

3.1.1 The Circuits

A description of how the circuits are connected according to the circuit diagram in Appendix XXX, follows below.

3.1.1.1 The Microcontroller

Despite of the differences in the shapes between the microcontroller on the wired prototype and the one on the circuit board the pins are connected in the same way and as follows:

Port A: the pins PA0-PA2 are used for control communication between the microcontroller and the TTS circuit. PA0, configured as an output, is connected to the SS pin on the TTS circuit and is used for initiating SPI transmission. PA1 is an input connected to the RB pin on the TTS circuit and indicates if the TTS circuit is ready to accept commands or not. The pin PA2 is used for resetting the TTS circuit and is therefore connected to its RESET pin. PA3 is connected to the SD SEL pin on the amplifier and is used for turning the amplifier on.

Port B: the pins PB5-PB7 are used for SPI and are connected to the corresponding pins either on the TTS circuit or to the flash socket used by AVRISP.

Port C: since no external memory is needed, port C is not used.

Port D: pin PD0 is the UART input line, used for communication with the phone, and is connected to the CFMS pin on the system connector. PD2 and PD3 serve as external interrupts and are used for volume control.

Pin XTAL1 and XTAL2 are the input to and output from the inverting oscillator amplifier and to the internal clock operating circuit. A crystal with a frequency of 4MHz is connected, according to the datasheets.
The RESET pin on the microcontroller is connected either to the AVRISP or supply voltage via the flash socket.

GND is connected to ground and VCC to supply voltage i.e. +2.7V.

The remaining pins are not used in this application.

When both the hardware and the software behaved as expected the emulator was changed to a real hole mounted microcontroller. Since the emulator has a pin amount of 40 pins, it must be exchanged to a microcontroller of the same shape. For this task a microcontroller with a 40 pins DIL package was used. This is not the same package intended to be used in the final circuit board, for this a 44 pins TQFP was used. The microcontrollers have exactly the same behavior; the four extra pins are of no use, i.e. not connected.

3.1.1.2 The Text-to-Speech Circuit

The TTS circuit used in the prototype is exactly the same that is used in the final circuit board which means that it is surface mounted. To be able to connect the circuit to the wired board a special socket must be used. The socket converts a surface mounted circuit to a hole mounted. Since the TTS circuit has 56 pins which is a unusual number, the socket was special ordered, and to be able to connect the socket a special circuit board must be created. The circuit board translates the pin configuration to a shape that fits the prototype board. FIGURE

Apart from the pins connected to the microcontroller mentioned in chapter 3.1.1, are the rest of the pins connected as follows:

The pins VSSA, VSSD are connected directly to ground while the pins AUXIN (according to data sheet) and CS (sets the TTS circuit to slave) are connected to ground via pull-down resistors.

The pins VCCD and VCCA are connected to supply voltage.

The pins XTAL1 and XTAL2 are connected to a crystal with frequency 24.576MHz to ensure correct functionality.

The SP+ and SP- pins are connected to the amplifier IN+ and IN- pins.

3.1.1.3 The Amplifier

Since both the mounted loudspeaker and the headphone needed amplification it seemed natural to connect the amplifier directly after the TTS circuit. The IN+ and IN- pins are connected as described in chapter 3.1.2 and the rest of the pins are connected as follows:

In this application the SD MODE pin is connected to supply voltage, and if the SD SEL pin, connected to PA3 on the microcontroller as described in chapter 3.1.1, is held low the amplifier will be on all the time.

The pin VDD is connected to the system connectors DCIO pin and GND and BYPASS are connected to ground.

The VO1 and VO2 pins are connected to the tele socket. These pins are also connected to the IN+ respective IN- through feedback resistors. To achieve a Gain=2 R_F must be twice the value of R_I, see chapter 2.1.3.

3.1.1.4 The Voltage Regulator

The voltage regulator was not connected on the wired board until the microcontroller and the TTS circuit communicated properly. On the other hand it was the first component connected on the final circuit board to ensure correct voltage supply.

The pins IN and SHDN are connected to the DCIO pin on the system connector, and GND is connected to ground. The OUT and SET pins are connected to a voltage divider that sets the output voltage level used as the supply voltage for the rest of the accessory. The divider consists of two resistors R_1 and R_2 mentioned in chapter 2.1.4. To achieve a supply voltage of +2.7V is R_1 calculated to 116kΩ. To be sure that the voltage level is above +2.7V, R_1 is set to 120 kΩ.

3.1.1.5 The Buttons

The buttons are connected to PD2 and PD3 i.e. the external interrupt pins on the microcontroller, chapter 3.1.1. Since these pins have internal pull-up resistors a press on the buttons will generate a low pulse on the pins.

3.1.1.6 The System Connector

To set the phone in service mode, VPPFLASH on the system connector must be supplied with at least +3V. This is achieved by connecting it to DCIO. The data to be sent from the phone is put on CFMS and this pin is connected to the UART input line on the microcontroller, PD0. The DCIO is the input voltage to the voltage regulator and DGND connects the phone ground to the accessory ground.

3.1.2 The Circuit Board

The final step was to develop the accessory in its true shape. This was done in the following steps:

1. A circuit diagram, showing the couplings between the circuits, was drawn.
2. A placement drawing, showing the physical placement of the circuits, and the template for the Printed Circuit Board, PCB, was created.
3. The PCB was etched with the pattern created in step 2.
4. The Printed Board Assembly, PBA, was created by soldering the components on the PCB.

The circuit diagram, the placement drawing and the template was created in Easy-PC Number One System which is an electrical CAD program. To make the accessory as small as possible, the PCB and the PBA was made double sided i.e. components on both sides.

Figure 3.1 The front and back side of the PCB.
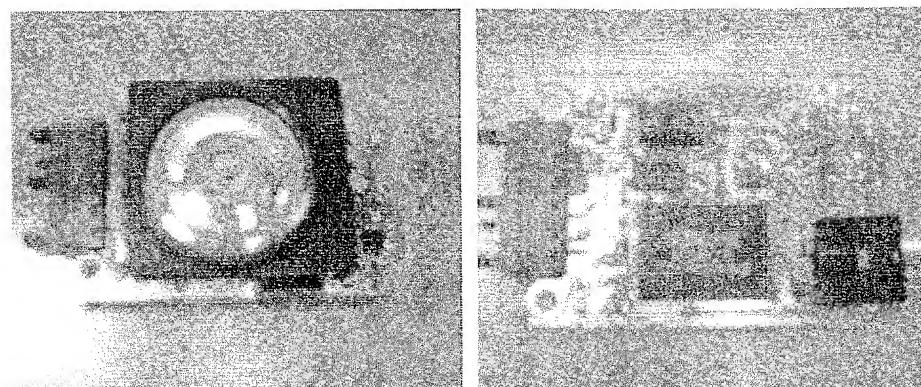
Figure 3.2 The front and back side of the PBA.

3.1.2.1 Etching and Soldering

The template created in Number One System, Appendix XXX, may be printed on transparent plastic and placed on a photo resist copperplate. The plate must then be exposed to UV-rays to make the photo resist stick to the copper. Then the plate must be dipped in a developer for positive 20 and thereafter placed in an etching bath. After the bath the plate should be washed with water and finally wiped with acetone.

When making the PCB board to the accessory, the plate was exposed for approximately 6 minutes and was etched for about 10 minutes. It is not possible to define a precise time in this process. It is a work of feeling. The different steps in the etching process can be seen in the Figure 3.3-3.9.

Figure 3.3 The copper plate.
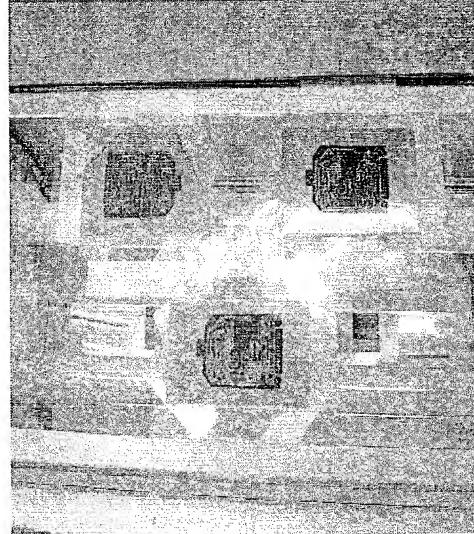
Figure 3.4 The plates with the pattern to be exposed.
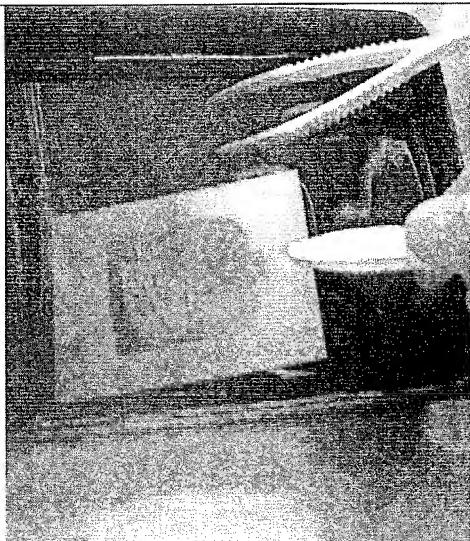
Figure 3.5 The plate in the developer.
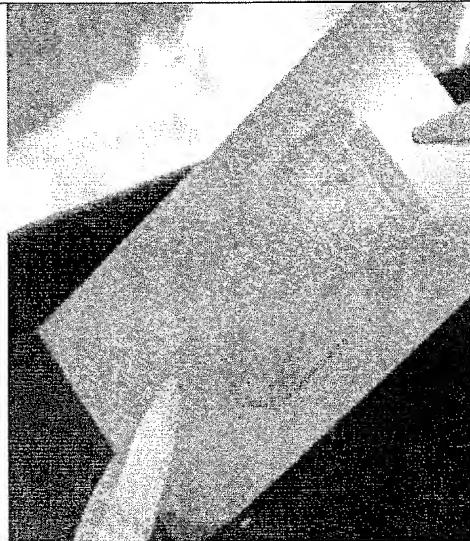
Figure 3.6 The developed plate with the pattern visible.
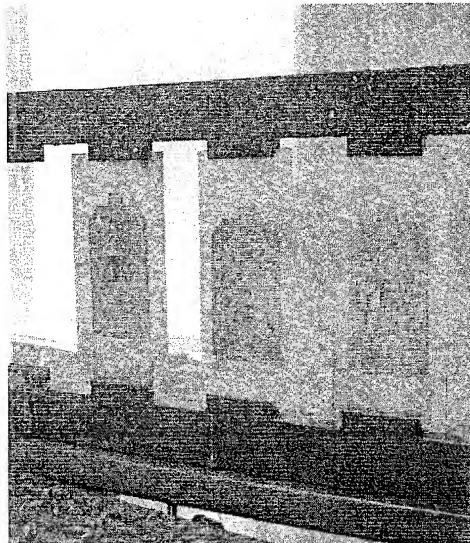
Figure 3.7 The etched plates.
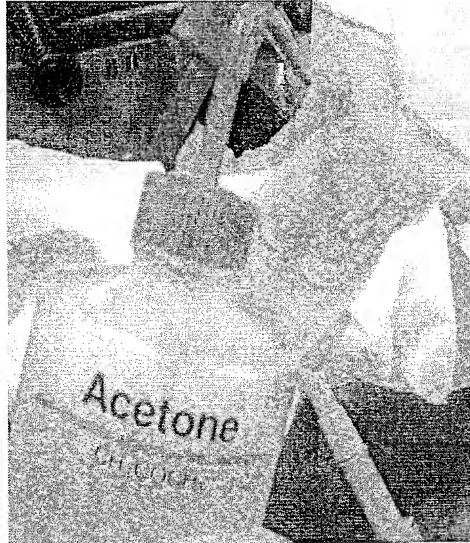
Figure 3.8 The plate being wiped with acetone.
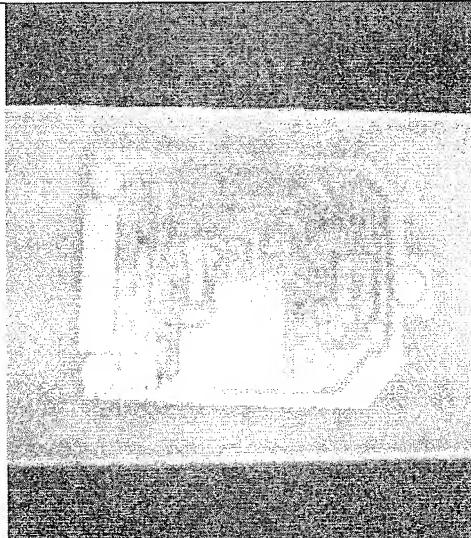
Figure 3.9 The final PCB.

The PCB was trimmed to its final shape by cutting of the superfluous plastic. To connect the both sides of the plate, appropriate holes were drilled and filled with conductive material. Then the components were soldered on their right positions and the PBA was built in its aluminum box. Finally the microcontroller was flashed with the release SOFTWARE and the MS with its specific SOFTWARE.

3.2 The Software

3.2.1 The Microcontroller Software

The software consists of three files; *TTS.c, int_routines.c* and *SPI_com.c*, see Appendix XXXX. All code intended for the microcontroller is compiled by IAR Embedded Workbench, chapter XXXX, and executed in AVR Studio, chapter XXXX, with the emulator ICE200. After having a well-functioning code, the emulator is changed to a microcontroller which is flashed by AVR In-System Programmer (ISP), chapter XXXX.

3.2.1.1 TTS.c

In TTS.c both the microcontroller and the TTS circuit are initialized. The microcontroller's registers are set to ensure wanted functionality, see Appendix XXX. The TTS circuit is initialized by sending the right commands in the right order to it. These commands powers up, starts the crystal, sets the speech pitch etc.

The TTS.c is the main program and contains the following functions:

void main(void); the main function with an infinite while-loop waiting for incoming data and button presses.

```
┌──────────┐
│  main(); │
└──────────┘
```

Is the entire word received? — YES → Is the TTS circuit converting? — YES → Finish conversion

Is the TTS circuit converting? — NO

Is the input buffer of the TTS circuit full? — NO → Start conversion

Is the input buffer of the TTS circuit full? — YES → Read the status of the TTS circuit

Increase volume? — YES → Is the max volume set? — NO → Increase volume

Increase volume? — NO

Is the max volume set? — YES → No action

Decrease volume? — YES → Is the min volume set? — NO → Decrease volume

Is the min volume set? — YES → No action
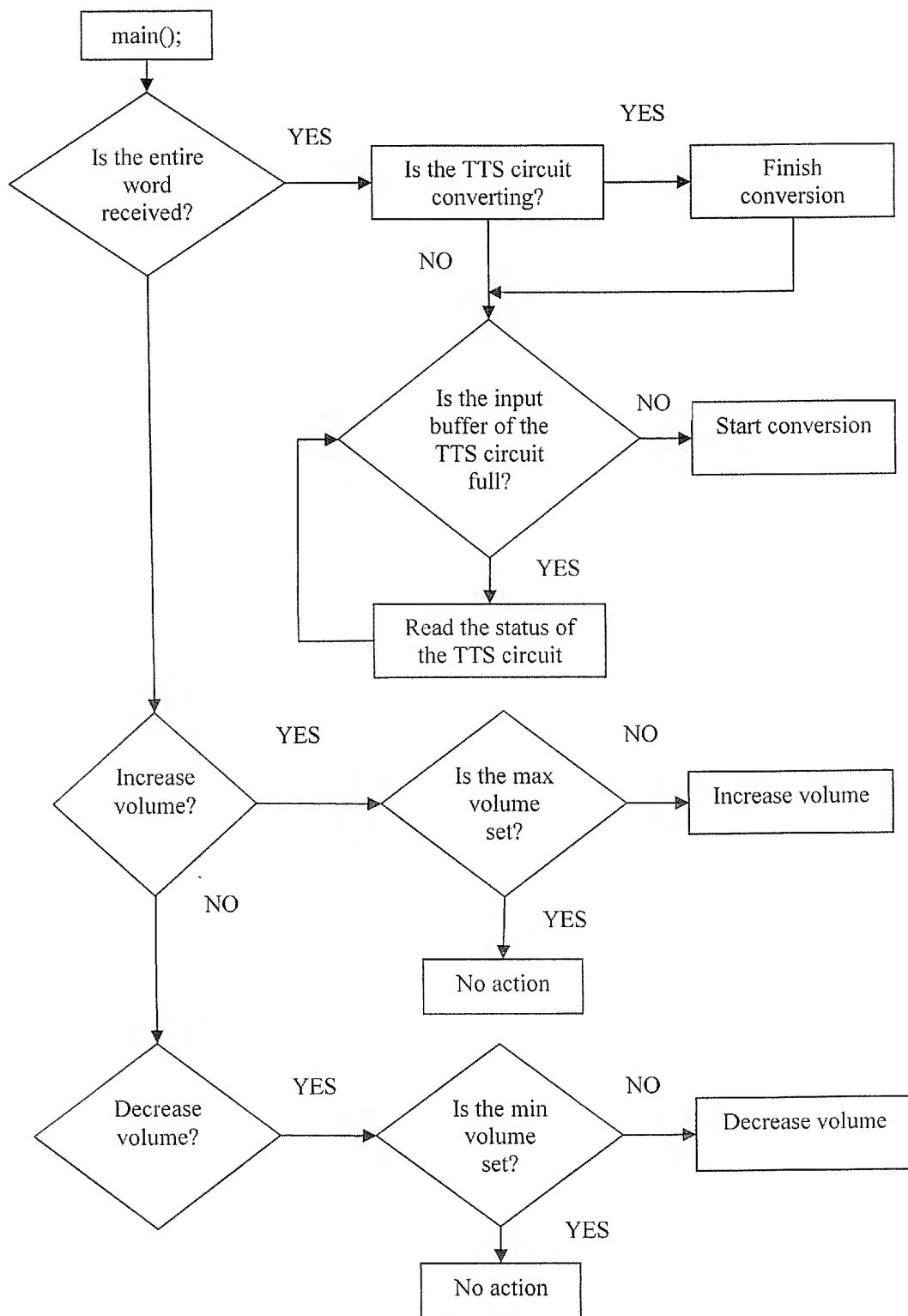
Figure 3.10 Flow chart over the *main()* function.

void init(void); initiates the microcontroller to wanted functionality by setting the ports to either inputs or outputs and by setting the registers.

void init_tts(void); initiates the TTS circuit by resetting it and sending start-up commands in a specific order according to the data sheets, see the code in Appendix XXX.

void reset(void); resets the TTS circuit (hardware reset).

3.2.1.2 int_routines.c

In int_routines.c the external interrupts and data communication interrupts are handled. The microcontroller provides 12 different interrupt sources and each interrupt has a separate program vector in the program memory space. The interrupts have different priority levels and the lower the address the higher the priority. Each address source can be reached with a specific name indicating the interrupt referred to.

All the possible interrupts are not used in this application. The ones used are the following:

interrupt [INT0_vect] void INT0_interrupt(void); interrupt routine for the external interrupt that increases the volume.

interrupt [INT1_vect] void INT1_interrupt(void); interrupt routine for the external interrupt that decreases the volume.
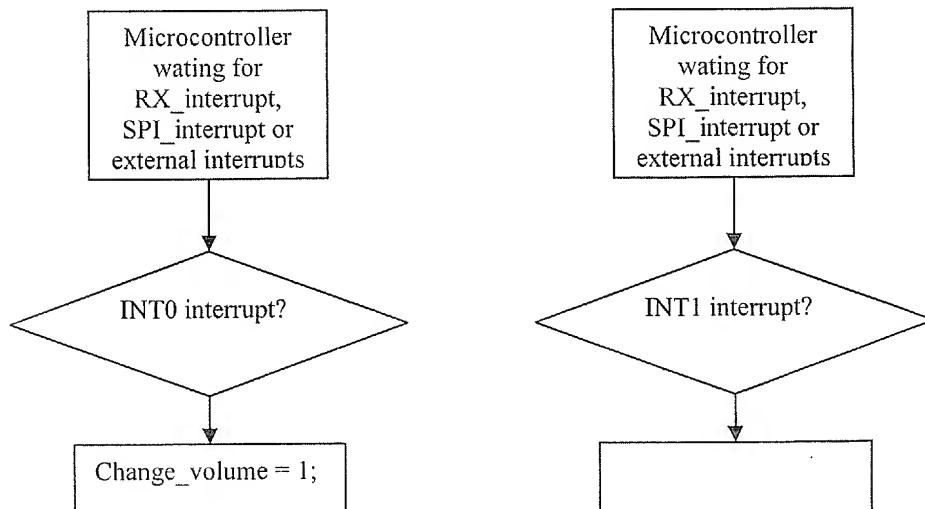
Figure 3.11 Flow chart over the external interrupt functions.

interrupt [SPI_STC_vect] **void** *SPI_STC_interrupt(void);* interrupt routine for reading status bytes from the TTS circuit, occurs when serial transfer is completed.

```
                    ┌─────────────────────┐
                    │    Microcontroller  │
                    │      waiting for    │
                    │     RX_interrupt,   │
                    │    SPI_interrupt or │
                    │  external interrupts│
                    └─────────────────────┘
                              │
                              ▼
                          ╱       ╲
                        ╱    SPI    ╲
                        ╲ interrupt ? ╱
                          ╲       ╱
                              │ YES
                              ▼
                          ╱       ╲          NO         ╱       ╲
                        ╱ status_nbr=1 ╲──────────────╱ status_nbr=2 ╲
                        ╲     ?      ╱               ╲      ?     ╱
                          ╲       ╱                   ╲       ╱
                              │ YES                        │
                              ▼                            ▼
              ┌───────────────────────┐      ┌───────────────────────┐
              │  Read the status from │      │  Read the status from │
              │       the TTS circuit │      │     the TTS circuit   │
              │   tts_status0=SPDR;   │      │   tts_status1=SPDR;   │
              └───────────────────────┘      └───────────────────────┘
                              │                            │
                              ▼                            │
                    ┌─────────────────┐◄──────────────────┘
                    │   status_nbr=0  │
                    └─────────────────┘
```
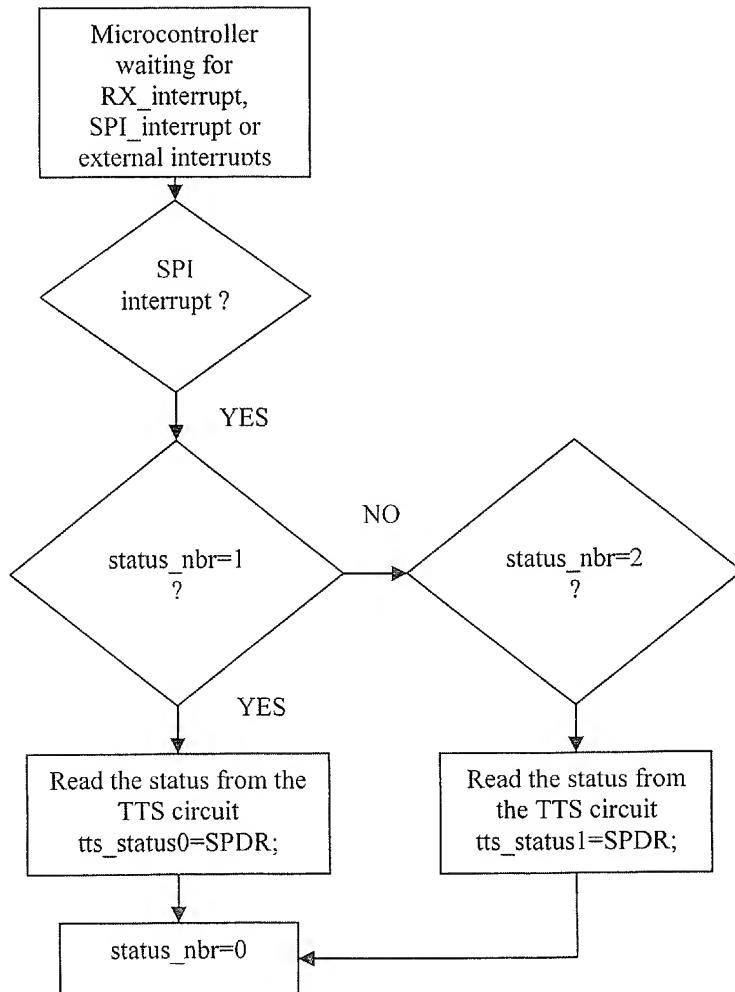
Figure 3.12 Flow chart over the SPI interrupt.

interrupt [UART_RX_vect] **void** *UART_RX_interrupt(void);* interrupt routine executed when character from the phone is received.

Microcontroller waiting for
RX_interrupt, SPI_interrupt or
external interrupts

If
RX_interrupt

YES

NO

FE=0
No errors

Read the
UDR register

YES

Save the received
character;
received _char=UDR;

YES

Ready to
receive
characters?

chars_for_conversion[
nbr_of_chars]
=received_char;
nbr_of_chars++;

NO

YES

Were the
three previous
characters #?

chars_for_conversion[0]
=received_char;
nbr_of_chars=1;
ready to receive=true:

NO

YES

Were the
previous and
the received
characters %?

Delete the last two
characters in
chars_for_conversion[]
ready_to_convert=true;

Update the values of the
previous received characters.

Figure 3.13 Flow chart over the RX interrupt.

3.2.1.3 SPI_com.c

The SPI.c contains the needed functions for communication between the microcontroller and the TTS circuit. To initiate a transfer the SS pin on the TTS circuit must go from high to low condition and to finish a transfer it must go high again.

The following functions are needed for SPI communication:

void send_cmd(char cmd_byte, char cmd_data_byte); used for sending commands to the TTS circuit.

void send_proc(char cmd_byte, char cmd_data_byte); used for shifting command bytes to the TTS circuit.

send_cmd();

Read status from TTS circuit

Is the TTS circuit ready to accept commands?

NO

YES

Send_proc(cmd1, cmd2);

Does the TTS circuit ignore commands?

YES

send_proc();

Set SS=1 and then SS=0 to initiate SPI transaction

Send the first command byte, cmd1

Set SS=1 to end SPI transaction
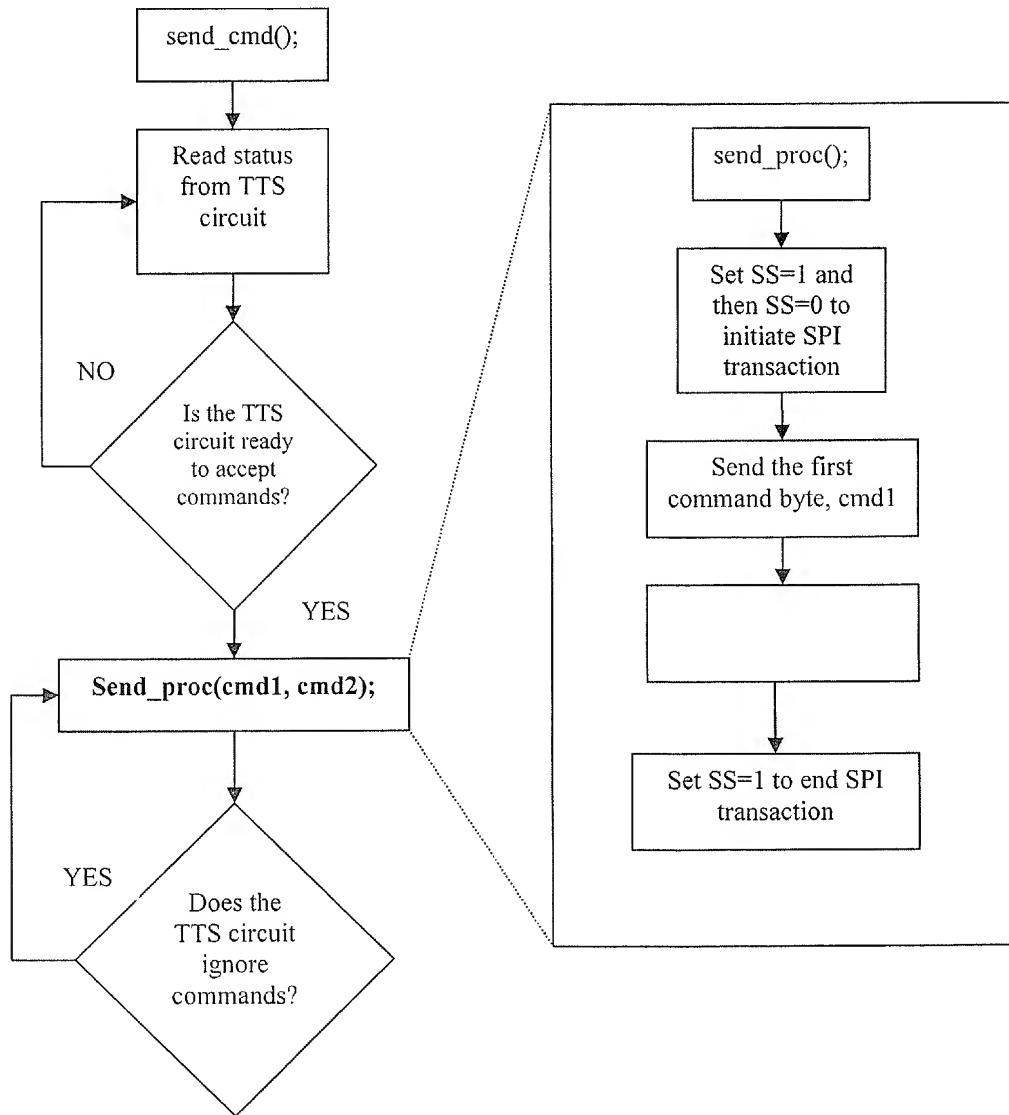
Figure 3.14 Flow chart over the functions *send_cmd()* and *send_proc()*.

void convert(void); used for sending data to the TTS circuit.

void send_data(void); used for shifting data bytes to the TTS circuit.
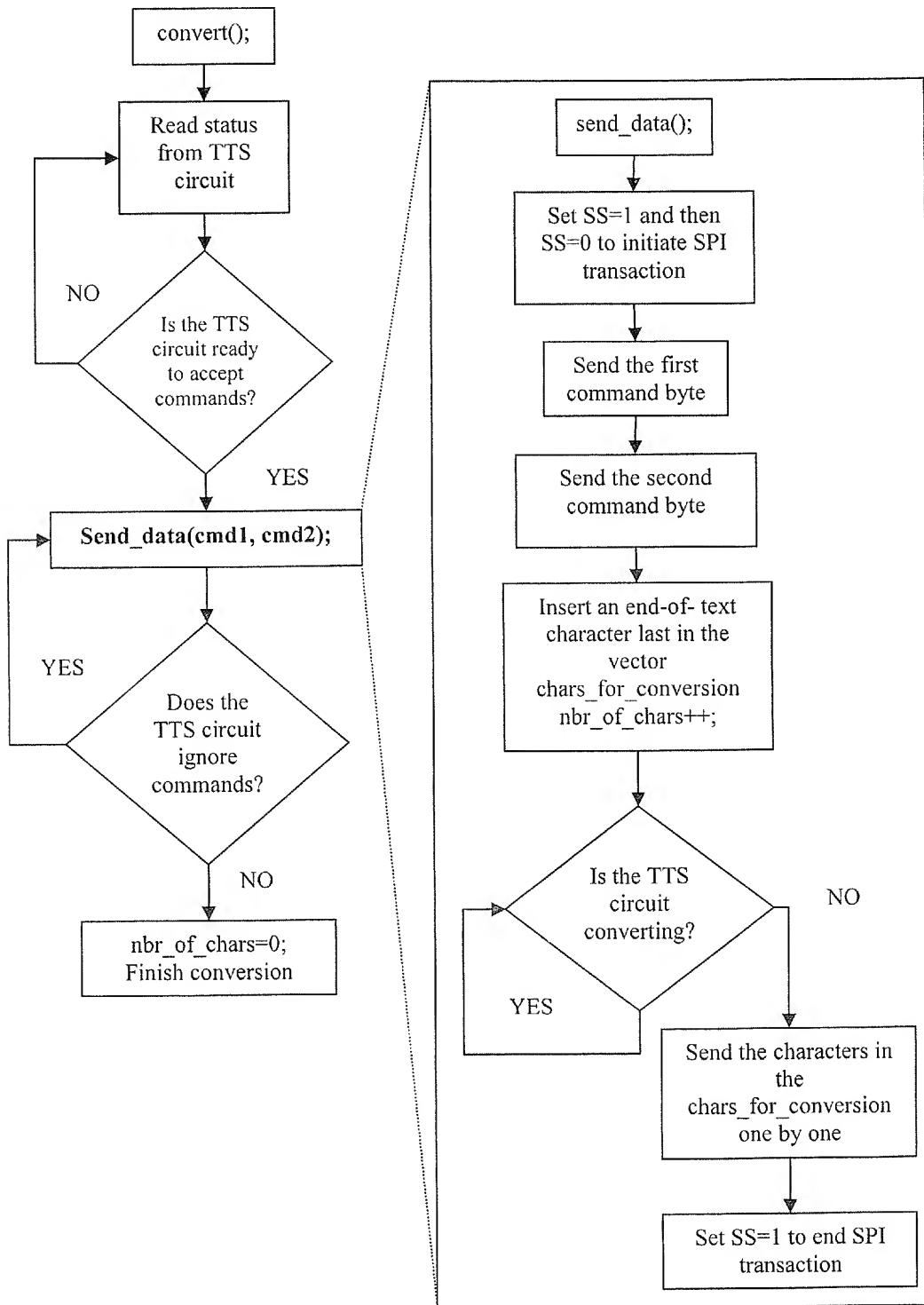
Figure 3.15 Flow chart over the functions *convert()* and *send_data()*.

The three files mentioned above and how their different functions interact with each other is shown in Figure 3.16. The function pointed at is the function called from the one pointing.
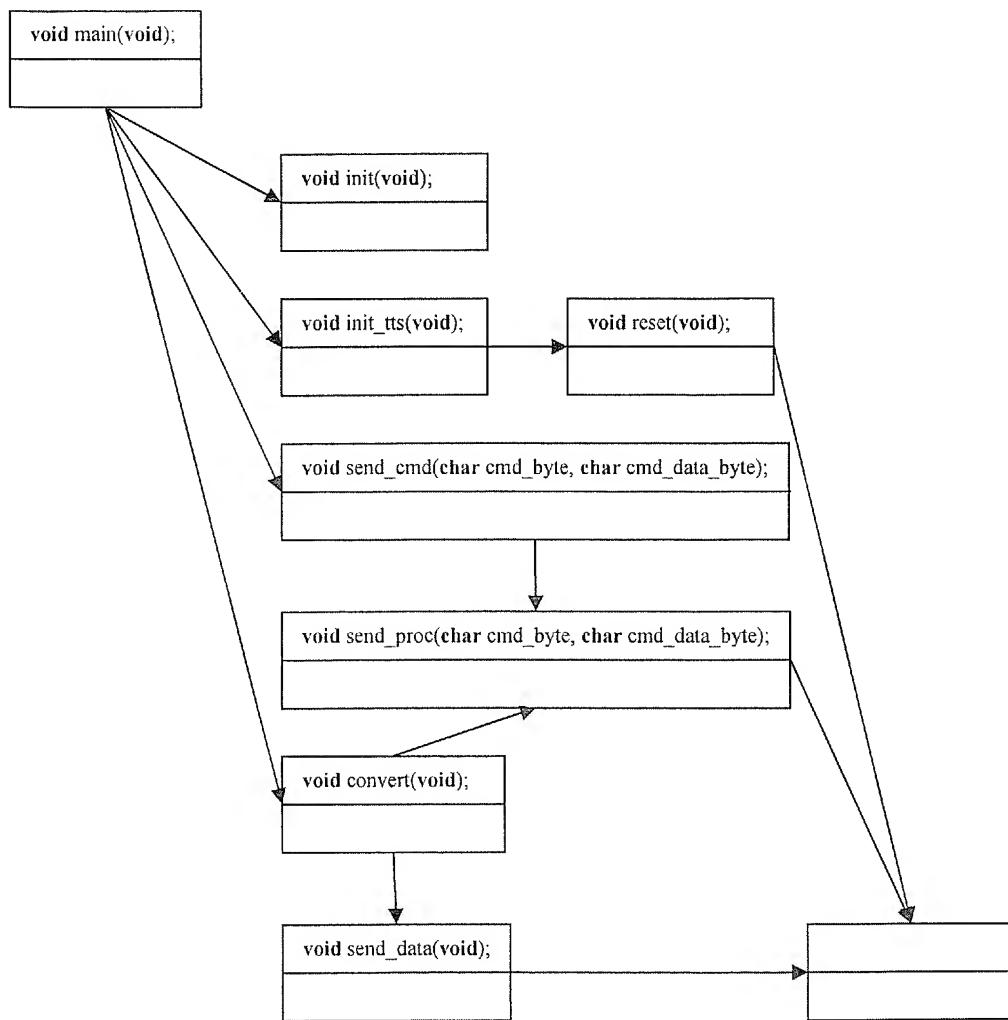
Figure 3.16 Block diagram over the interaction between the functions.

3.2.2 The Phone Software

All phone software is written in Visual C++ and is configuration managed by ClearCase and CME2.

3.2.2.1 The Changes in Software

A similar code as below was added to the modules with differences like the names of the text ids and how to reach them.

length = TextGetLength(item_p->textId);

```
buf = ALLOC_OBJMMI_DATA(WCHAR_t, (length+1));
label = ALLOC_OBJMMI_DATA(unsigned char, (length+1));
bufSize = TextGet((item_p->textId), buf, (length+1));
wstrntostr(label, buf, bufSize);
printf("\n%s%s%s\n", "###", label, "%%");
FREE_OBJMMI_DATA(&buf);
FREE_OBJMMI_DATA(&label);
```

The affected modules and files are:

- BA_SoftwareModule_SMS\cnh1010079_uimessaging\internsoftware\
    - sms_read.c: ReadMsg_Enter_Action(....){
- LD_SoftwareModules_002\cnh1010012_userint\internsoftware\disp\objects\navspace\
    - dsp_basl_laf.c: BaseListLaF_Impl_DrawItem(....){
    - dsp_desktop_laf.c: DesktopLaF_Impl_DrawItem(....){
- LD_SoftwareModules_002\cnh1010012_userint\internsoftware\disp\objects\
    - dsp_1ofm.c: OneOfManySelectionShowLabel(....){
    - dsp_nofm.c: NoOfManySelectionShowLabel(....){
    - dsp_onof.c: OnOffInputDraw(....){
    - dsp_feedback.c: FeedbackDraw(....){

To be able to call the function *wstrntostr(label, buf, bufSize);* the file *error_r_txt_ui.h* has to be included in the files above.

The MS transmits data through its system contact with a default baud rate of 115200bps. The microcontroller in the accessory achieves the lowest possible baud rate error at 9600bps and therefore the baud rate is set to this level both in the microcontroller and in the MS. The MS baud rate can be changed in the file *hardware.h*:

```
#define UART_MAIN_CTRL_BAUD_RATE_115200          (0x00)
#define UART_MAIN_CTRL_BAUD_RATE_9600            (0x00)
```

The hex value for *UART_MAIN_CTRL_BAUD_RATE_115200* is default 0xXX and has to be changed to 0x00.

When the MS is in service mode it constantly transmits data such as radio parameters on its system bus. Since the microcontroller handles all data that it receives on its UART it can easily become overloaded. This is solved by disabling the unnecessary prints, all of them except for the ones at the MS start up process. This is done in the file in *LD_Products_001\crh1069019_mia\DescrExtra.cfg* by writing XXXXXXXXXXXXXX.

4 The Tools

The tool used for drawing the circuit diagram and the placement diagram in this project was Number One System.

Developing the microcontroller software required several tools. IAR Embedded Workbench was used for compiling the software and AVR Studio version 4.0 was used for executing it. These were used together with the emulator ATICE200 version 1.10 and finally AVRISP was used for programming the microcontroller. During programming the TTS circuit must be disconnected, since it also uses the microcontroller's SPI interface. This is achieved by switching the coupling manually, see Appendix XXXX.

ClearCase and CME2 are used by Sony Ericsson Mobile Communications for handling the different versions of the phone software and therefore also used in this project.

4.1 Easy-PC Number One System

Easy-PC Number One System is an electrical CAD program used for creating circuit diagrams, placement diagrams and templates for Printed Circuit Boards, PCBs. The program contains a number of components in its default library, but the user has the opportunity to create new ones when needed. To do this, first the schematic symbol must be created. The symbol does not have to look exactly as the specific circuit but it is important that it has the right number of pins. To every schematic symbol a corresponding PCB symbol must be created. The PCB symbol must have the exact dimensions as the actual circuit, concerning both shape and pin layout. When both the schematic and the PCB symbol have been created the pins must be matched, meaning that the symbolic pin numbers are paired with the corresponding numbers on the PCB symbol.

When all the needed symbols are available the circuit diagram can be drawn, see Appendix XXXX. When this is done the program can easily (by a button press) translate it to the template for the PCB. The placement drawing, see Appendix XX, can then be created by manually placing the components to the wanted positions on the PCB.

4.2 IAR Embedded Workbench

IAR Embedded Workbench provides a powerful environment for developing projects with a range of different target processors. It has been specially designed to fit in with the way the software development projects are typically organized. It allows projects to be organized in a hierarchical tree structure showing the dependency between files at a glance.

4.3 Atmel AVR Studio

AVR Studio is used for execution of program code. It is an integrated development environment for writing and debugging AVR applications. It combines an editor, project manager, assembler/compiler interface and debugger in one development tool. The user can manage projects, edit source code, simulate or interface an emulator.

AVR Studio can be used together with IAR in such way that after writing and compiling the code in IAR, the compiled code can be opened in AVR Studio. If an emulator is detected on the serial port, AVR Studio automatically enters emulation mode, otherwise it starts in simulation mode. The program detects the target system and downloads the code to it. When the program is downloaded the user can set an unlimited number of breakpoints and can also watch the registers, SRAM, EEPROM, Flash and I/O as they change their values.

4.4 Atmel ATICE200

ATICE200 is an in-circuit emulator. This is a kit that contains several personality adapters with different numbers of pins for use in different purposes. Each adapter corresponds to one or two of Atmel's processors in the AT90S-series. The adaptor used in this project is the Atadap3000 that corresponds to AT90S8515. The ICE200 provides a debugging platform with a minimum of differences between the emulator and the actual processor it is emulating. This provides identical electrical characteristics. Figure 4.1 shows the emulator with its adaptor connected to the wired prototype.
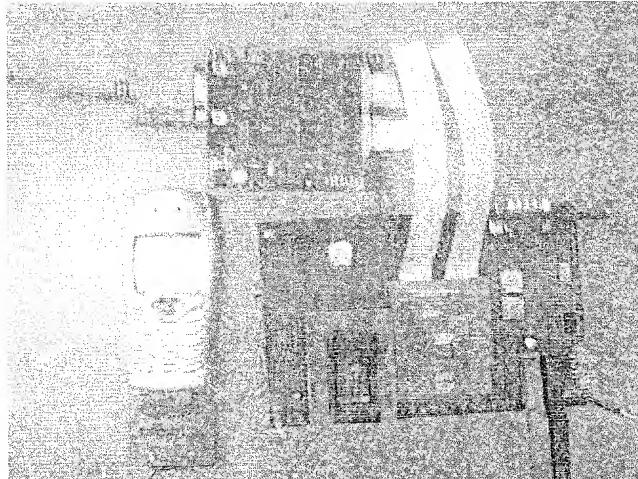

Figure 4.1 The ICE200, the MS and the wired prototype.

When used with the AVR Studio debugging environment, the ICE200 gives the user full run time control, unlimited number of breakpoints, symbolic debugging and full memory and register visibility.

The main board contains the program memory that holds the application code that is being emulated. It also contains logic for communicating with the host PC, and the breakpoint logic. The pod contains the AVR emulator chip and the personality adapters map the pin out from ICE200 pod to each microcontroller it supports. The adapter includes an identification code that AVR Studio uses for automatic device type detection. These parts are connected together with the target application.

4.5 Atmel AVRISP

AVR In-System Programmer, AVRISP, is used for flashing AVR microcontrollers. The programmer connects to a PC through a standard RS232 serial interface and uses AVR Studio as front-end software. The programmer uses the SPI interface to flash the microcontroller.
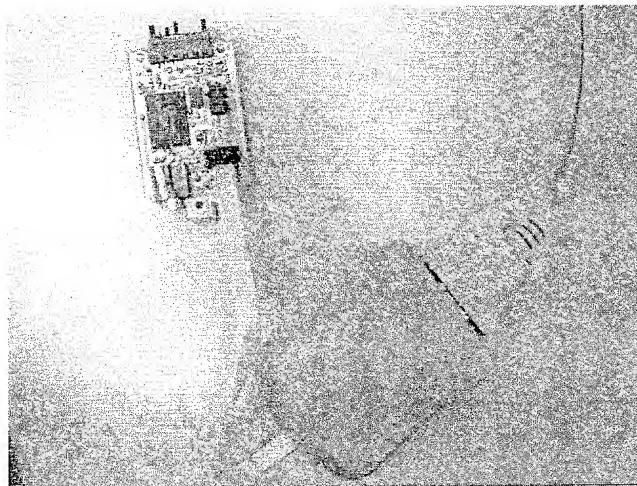
Figure 4.2 AVRISP connected to the PBA and to a computer via a RS232 cable.

4.6 ClearCase and CME2

ClearCase and CME2 are two programs used for software configuration management system that helps automate the tasks required to write, release, and maintain software. They track changes to every file and directory in the projects and supports parallel development.

5 Result

The purpose of this thesis was to develop a demonstrator that could read SMS and menus. The result was a well functioning accessory that fulfills the specified behavior. Both the hardware and the software works fine with just the few limitations discussed in chapter 5.2. Since this application is useful not only for the visually impaired but also for example as a complement for vehicle hands free, future development is not excluded.

5.1 How to use the accessory

To be able to use the accessory a Sony Ericsson T68i with special software is required.

- Make sure that the battery is loaded and that the phone is turned off.
- Attach the accessory to the phone.
- Turn on the phone; the accessory is ready for use.
- To change volume, pull the button upwards or downwards.
- To use headphones insert the connector in the tele socket.
- Before removing the accessory, turn off the phone.

5.2 Limitations

There are a few limitations that have to be considered when using the accessory:

- The accessory can not be attached to any MS. The MS must have the special SOFTWARE mentioned above.
- The accessory must be attached to the MS before the MS is turned on, to secure that the MS is in service mode.
- Since it was not possible to communicate with AT commands only the menus in the modified modules are read.
- When browsing fast the TTS circuit does not manage to handle all data sent to it since it turns off reception while converting.
- If removed when MS is on the MS will hang.
- The power consumption is rather high and a battery won't last for long.

6 Discussion

For future development it is not necessary to restrict to a hardware solution in form of an accessory as in this case. It is fully possible to build in the corresponding solution in the phone, either as an ASIC or as a pure software solution, considering that the phones today already contain everything needed. Since the interest for TTS is large in many places, this feels to be right in time.

To offer TTS in the MS is not only an aid for the visually impaired and the drivers but also a step further in personalizing the phone. Today personalization can be done by changing the panels, having different polyphonic sounds, the possibility to add pictures etc. Just imagine how it would be to have your favorite artist reading your SMS, by combining TTS with MIDI files. This brings whole new possibilities to multimedia, for example games.

The TTS can also be considered in combination with WAP and the Yellow Pages route service or with a future GPS function in the phone. Together with memory sticks it can be possible to have your favorite books read while sitting in the car, on the train or just relaxing in the hammock. This is only a small fraction of all possible features and the imagination is the only limit.

Bibliography

[1]   IAR Systems, EWA90-1, *AT90S Embedded Workbench Interface Guide for Atmel's AT90S Microcontroller Family*, September 1996

[2]   IAR Systems, ICCA90-1, *AT90S C Compiler Programming Guide for Atmel's AT90S Microcontroller Family*, September 1996

[3]   Number One Systems, *Easy-PC for Windows*, September 2000

[4]   Atmel, *8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash AT90S8515*, Rev 0841G-09/01, www.atmel.com

[5]   Winbond, *WTS701 Winbond Single-Chip Text-To-Speech Processor*, Rev 3.07, www.winbond-usa.com

[6]   Atmel, *AVR ICE200 User Guide*, 1999, www.atmel.com

[7]   National Semiconductor, *LM4894 Boomer*, June 2001, www.national.com

[8]   Maxim, *MAX8863*, www.elfa.se

[9]   Ericsson, *System Bus Specification for the Digital System Phones, CT/UT 93:003*, 1995-06-14

[10]   Ericsson, *Description of System Connector Electrical Interfaces for the Marianne Product Platform, TX/B 97:0028*, 2000-08-10

Appendix C – Register settings of AT90S8515

SREG – Status Register

I	T	H	S	V	N	Z	C
1	0	0	0	0	0	0	0

I – Global Interrupt Enable: Is set to enable global interrupts.
T – Bit Copy Storage: Not used in this application. Initial value is set to zero.
H – Half-carry Flag: Not used in this application. Initial value is set to zero.
S – Sign Bit: Not used in this application. Initial value is set to zero.
V – Two's Complement Overflow Flag: Not used in this application. Initial value is set to zero.
N – Negative Flag: Not used in this application. Initial value is set to zero.
Z – Zero Flag: Not used in this application. Initial value is set to zero.
C – Carry Flag: Not used in this application. Initial value is set to zero.

SP – Stack Pointer

Not used in this application.

GIMSK – General Interrupt Mask Register

INT1	INT0	-	-	-	-	-	-
1	1	-	-	-	-	-	-

INT1 – External Interrupt Request 1 Enable: Set together with SREG:I enables external interrupts. Decrease the volume.
INT0 – External Interrupt Request 0 Enable: Set together with SREG:I enables external interrupts. Increase the volume.
The rest of the bits are reserved.

GIFR – General Interrupt Flag Register

INTF1	INTF0	-	-	-	-	-	-
1	1	-	-	-	-	-	-

INTF1 – External Interrupt Flag 1: Is set to clear interrupt flag.
INTF0 – External Interrupt Flag 1: Is set to clear interrupt flag.
The rest of the bits are reserved.

TIMSK – Timer/Counter Interrupt Mask Register

TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	-
0	0	0	-	0	-	0	-

TOIE1 – Timer/Counter1 Overflow Interrupt Enable: Not used in this application. Initial value is set to zero.

OCIE1A – Timer/Counter1 Output CompareA Match Interrupt Enable: Not used in this application. Initial value is set to zero.

OCIE1B – Timer/Counter1 Output CompareB Match Interrupt Enable: Not used in this application. Initial value is set to zero.

TICIE1 – Timer/Counter1 Input Capture Interrupt Enable: Not used in this application. Initial value is set to zero.

TOIE0 – Timer/Counter0 Overflow Interrupt Enable: Not used in this application. Initial value is set to zero.

The rest of the bits are reserved.

TIFR – Timer/Counter Interrupt Flag Register

Not used in this application.

MCUCR – MCU Control Register

SRE	SRW	SE	SM	ISC11	ICS10	ISC01	ISC00
0	0	0	0	1	1	1	1

SRE – External SRAM Enable: Not used in this application. Initial value is set to zero.

SRW – External SRAM Wait State: Not used in this application. Initial value is set to zero.

SE – Sleep Enable: If set SLEEP is enabled. Is set just before execution.

SM – Sleep Mode: Is cleared to select Idle Mode.

ISC11 – Interrupt Sense Control 1, Bit 1: Is set to activate interrupt 1 at rising edge.

ISC10 – Interrupt Sense Control 1, Bit 0: Is set to activate interrupt 1 at rising edge.

ISC01 – Interrupt Sense Control 0, Bit 1: Is set to activate interrupt 0 at rising edge.

ISC00 – Interrupt Sense Control 0, Bit 0: Is set to activate interrupt 0 at rising edge.

TCCR0 – Timer/Counter0 Control Register

Not used in this application.

TCNT0 – Timer Counter0

Not used in this application.

TCCR1A – Timer/Counter1 Control Register A
Not used in this application.

TCCR1B – Timer/Counter1 Control Register B
Not used in this application.

TCNT1H and TCNT1L – Timer/Counter1
Not used in this application.

OCR1AH and OCR1AL – Timer/Counter1 Output Compare Register
Not used in this application.

OCR1BH and OCR1BL – Timer/Counter1 Output Compare Register
Not used in this application.

ICR1H and ICR1L – Timer/Counter1 Input Capture Register
Not used in this application.

WDTCR – Watchdog Timer Control Register
Not used in this application.

EEARH and EEARL – EEPROM Address Register
Not used in this application.

EEDR – EEPROM Data Register
Not used in this application.

EECR – EEPROM Control Register
Not used in this application.

SPCR – SPI Control Register

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
1	1	0	1	1	1	0	0

SPIE – SPI Interrupt Enable: Is set to enable SPI interrupts.

SPE – SPI Enable: Is set to enable SPI.
DORD – Data Order: The MSB of the data word is transmitted first when cleared.
MSTR – Master/Slave Select: Is set to select Master SPI Mode.
CPOL – Clock Polarity: Is set to make SCK high when idle.
CPHA – Clock Phase: Selectable.
SPR1 – SPI Clock Rate Select 1: Control the SCK rate of the master.
SPR0 – SPI Clock Rate Select 0: Control the SCK rate of the master.

SPSR – SPI Status Register

Read only.

SPDR – SPI Data Register

Writing to the register initiates a data transmission and will be set just before transmission.

UDR – UART I/O Data Register

It alternates between being a register storing incoming data and outgoing data.

USR – UART Status Register

RXC	TXC	UDRE	FE	OR	-	-	-
-	1	-	-	-	-	-	-

RXC – UART Receive Complete: Is set when received character is transferred from Receiver Shift register to UDR.
TXC – UART Transmit Complete: Is set when the entire character is shifted out from Transmit Shift register and no new data is written to UDR. Not used.
UDRE – UART Data Register Empty: Zero indicates that the transmitter is ready to receive a new character for transmission.
FE – Framing Error: Is set if a framing error is detected.
OR – Overrun: Is set if an overrun condition is detected.

UCR – UART Control Register

RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
1	0	0	1	0	0	-	-

RXCIE – RX Complete Interrupt Enable: Is set to enable the Receive Complete Interrupt routine.
TXCIE – TX Complete Interrupt Enable: Transmit Complete Interrupt routine is not used.

UDRIE – UART Data Register Empty Interrupt Enable: Enables the
UART Data Register Empty Interrupt routine if set.
RXEN – Receiver Enable: Is set to enable the UART receiver.
TXEN – Transmitter Enable: Is not used.
CHR9 – 9-bit Characters: Makes characters 9 bits long plus start and stop
bits if set.
RXB8 and TXB8 depends on CHR9, contain the ninth bit.

UBRR – UART BAUD Rate Register

MSB							LSB
0	0	0	1	1	0	0	1

UBRR set to 25 corresponds to a BAUD rate of 9600 when a 4 MHz
crystal is used.

ACSR – Analogue Comparator Control and Status Register

ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
1	-	-	0	0	0	0	0

ACD – Analogue Comparator Disable: Is set to switch of the power to the
Analogue Comparator.
The other bits are not used in this application.

DDRA – Port A Data Direction Register

DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
1	1	1	1	1	1	0	1

The Port A pins are set as outputs, except for pin A1.

DDRB – Port B Data Direction Register

SCK	MISO	MOSI	SS	AIN1	AIN0	T1	T0
1	0	1	1	1	1	1	1

SCK – Serial Clock: Master clock output.
MISO – Master In Slave Out: Master data input.
MOSI – Master Out Slave In: Master data output.
SS – Slave Select: Master select output.
AIN1 – Analogue Comparator Negative Input: Not used in this application.
AIN0 – Analogue Comparator Positive Input: Not used in this application.
T1 – Timer/Counter1 Counter Source: Not used in this application.
T0 – Timer/Counter0 Counter Source: Not used in this application.

DDRC – Port C Data Direction Register

DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
1	1	1	1	1	1	1	1

The Port C pins are set as outputs.

DDRD – Port D Data Direction Register

DDD7	DDD6	OC1A	DDD4	INT1	INT0	TXD	RXD
1	1	1	1	1	1	1	0

OC1A – Output Compare Match Output: Not used in this application.
INT1 – External Interrupt Source 1: Volume button.
INT0 – External Interrupt Source 0: Volume button.
TXD – Transmit Data: Is set to output.
RXD – Receive Data: Is set to input.
Other pins are not used in this application.

Appendix D – The Software

D.1 The C-files

D.2 The H-files